VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

ROZŠÍŘENÍ GRAFICKÝCH ROZHRANÍ ZVUKOVÝCH ZÁSUVNÝCH MODULŮ O 3D GRAFIKU
3D GRAPHICS EXTENSION OF GRAPHICAL USER INTERFACES OF AUDIO PLUGINS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Filip Dufka**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Frenštátský**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Filip Dufka ***ID:*** 174447
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Rozšíření grafických rozhraní zvukových zásuvných modulů o 3D grafiku

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je porovnat možnosti využití 3D grafiky v grafických rozhraních zvukových zásuvných modulů jako je technologie VST. Student porovná možnosti frameworků využívaných pro tvorbu grafických uživatelských rozhraní zvukových zásuvných modulů o implementaci 3D grafických objektů.
V rámci práce student vytvoří zásuvný modul techologie VST3, který umožní vytvářet komplexní grafickou scénu s možností uživatelské interakce.

DOPORUČENÁ LITERATURA:

[1] SYSEL, P.; SMÉKAL, Z. Číslicové filtry. Brno: Vysoké učení technické v Brně, 2012. s 1-148. ISBN 978-8--214-4454-6

[2] WHITROW, R. J. OpenGL graphics through applications. New York: Springer, 2008. ISBN 9781848000230.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Petr Frenštátský
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Tato práce se věnuje možnostem rozšíření grafického uživatelského rozhraní modulů VST3, které slouží pro digitální zpracování zvukového signálu, o implementaci 3D modelů s využitím OpenGL. Práce se zaměřuje na vytvoření algoritmu pro možnost změny polohy přicházejícího zvuku zapouzdřeného do zásuvného zvukového modulu s rozšířeným grafickým rozhraním obsahující 3D vizualizaci.

Klíčová slova

Digitální zpracování zvuku, grafické uživatelské rozhraní, OpenGL, VSTGUI, VST3

Abstract

This thesis deals with possibilities of an extension of audio modules graphical user interface with 3D models using OpenGL framework. As an audio module implementing digital signal processing algorithms Virtual Studio Technology (VST) is used. The objective of this thesis is a creation of an algorithm for a change of sound source location. The algorithm is implemented in VST3 with an additional 3D visualization.

Keywords

Digital audio processing, graphical user interface, OpenGL, VSTGUI, VST3

DUFKA, F. *Rozšíření grafických rozhraní zvukových zásuvných modulů o 3D grafiku.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 25 s.Vedoucí bakalářské práce Ing. Petr Frenštátský.

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Frenštátskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Také bych chtěl poděkovat kolektivu firmy Audified za poskytnutí verzovacího systému a podnětných připomínek.

V Brně dne ....

podpis autora

Obsah

Seznam obrázků

1 Úvod

Běžná praxe hudebního průmyslu probíhá dnes převážně digitálně na stolních počítačích. Zvukový signál se zde vytváří, nebo je sem přiveden z AD převodníků a je zde uchováván a zpracováván ve formě jednotlivých číselných vzorků. Sekvence těchto číselných vzorků představuje velikost fiktivního akustického tlaku v časových intervalech. Tyto časové intervaly jsou dány vzorkovací periodou, jejíž hodnota začíná na 22,7 μs, a s rostoucí kvalitou tento časový úsek klesá. Pokud je žádoucí, aby byl zvukový signál zpracováván aspoň tak rychle, jak má být následně slyšen, musí celý zvukový řetězec zvládnout zpracovat jednu číselnou hodnotu právě za tento, nebo nižší, čas.

Dnešní kvalita a rychlost stolních počítačů tuto podmínku s přehledem splňuje, a tudíž je možné provádět okamžité a nedestruktivní změny na zvukovém materiálu, což dříve nebylo možné. S touto technologickou výhodou rostou i možnosti zpracování zvukového signálu, kdy je možné využít velké množství zvukových algoritmů, efektů a generátorů. Pro úpravu zvukového signálu je žádoucí, aby si jednotlivé prvky zvukového řetězce mohl uživatel navolit sám, a proto je zde, stejně jako v analogovém prostředí, zvolen modulový přístup.

Práce při úpravě zvuku probíhá proto většinou v dedikované aplikaci, která umožňuje zpracovávat zvukový signál a posílat jej dál do zvukového ovladače stolního počítače. Tato aplikaci, které říkáme *Digital Audio Workstation* (DAW), umožňuje upravovat a míchat dohromady několik digitálních zvukových signálů. V této bakalářské práci bude tato aplikace často referovaná jako hostitelská aplikace, protože umožňuje spravovat a spouštět procesy *zvukových zásuvných modulů* – jednotlivých článků v řetězci zvukového signálu.

Konkrétní nastavení zvukového modulu je možné ovlivňovat pomocí parametrů. Tyto parametry může měnit pouze hostitelská aplikace pomocí speciálního okna – editoru, ve kterém má uživatel možnost přizpůsobit situaci jednotlivé parametry. Tento editor vytváří hostitelská aplikace sama. Často ale způsob, jak má editační okno vypadat, definuje právě zásuvný modul tak, aby umístění jednotlivých uživatelských prvků nejvíce vyhovovalo funkci zvukového zásuvného modulu. Je zde také možnost zpříjemnit koncovému uživateli práci umístěním vizualizace aktuálního stavu zvukového modulu. Odpovědi na otázky: *„jak kvalitně zvukový modul funguje a vypadá“*, jsou důležitým měřítkem při koupi a prodeji zvukového modulu.

Obecně platí, že čím komplexnější je vizuální stránka editoru, tím vyšší jsou požadavky na práci centrální procesní jednotky (CPU) stolního počítače. Vzhledem k faktu, že tento procesor má na starosti také zpracování samotného zvukového signálu, dělají se ústupky právě na této části zásuvného modulu, aby nedošlo ke zmenšení času určenému k výpočtu výsledné hodnoty zvukového signálu.

Stolní počítače mají v dnešní době přiřazenu speciální sadu procesorů, označovánu jako *grafická procesní jednotka* (GPU), která se stará pouze o grafický výstup počítače. Tato sada procesorů umožňuje paralelní zpracovávání dat pro grafické vykreslování barevné informace převážně 3D grafiky. V době, kdy dochází na centrální

procesní jednotce k výpočtu jednotlivých zvukových vzorků, grafická jednotka zahálí, a nevyužívá svůj značný výkon.

Je tedy logickým krokem přenést většinu úkonů spojených s grafickým uživatelským rozhraním do zodpovědnosti grafického akcelerátoru. Protože se o vykreslování stará hostitelská aplikace, je nutné zajistit komunikaci mezi zásuvným modulem a grafickou kartou. Prozkoumání této problematiky je přesně cílem této bakalářské práce. Výstupy pak budou ověřeny vytvořením zvukového zásuvného modulu VST3 s komplexním řešením grafického uživatelského rozhraní.

2 Zvukové zásuvné moduly

Zvukový zásuvný modul (*plug-in*) je soubor algoritmů sloužící k úpravě, analýze nebo generování zvukových signálů. Algoritmy jsou implementovány ve formě dynamických knihoven, které je možné vkládat do *hostitelských aplikací*, které přes dané rozhraní přistupují k funkcionalitám knihovny. Jedná se o digitální obdobu analogových zvukových procesorů.

Pokud modul zvukový signál upravuje, hovoříme o něm také jako o **zvukovém efektu**. Sem patří i zvukové analyzátory, které signál neovlivňují, ale pouze ho využívají k výpočtu nebo vizualizaci.

V případě, že modul zvukový signál vytváří, jedná se o **virtuální hudební nástroj**. Jeho činnost je ovládána převážně MIDI zprávami, které mohou být zaznamenány do stopy, nebo je lze posílat v reálném čase.

V rámci hostitelské aplikace lze na jednotlivé stopy a sběrnice vkládat zvukové moduly. Hostitelská aplikace nemá přímou kontrolu nad zvukovým modulem, ale zná základní informace, které jsou nutné k chodu pluginu. Mezi tyto informace patří například počet vstupních a výstupních kanálů nebo výčet parametrů.

2.1 Parametry

Parametry jsou proměnné, ke kterým má přístup jak hostitelská aplikace, tak samotný zvukový modul. K nastavení parametru slouží primárně uživatelské prostředí pluginu, ale lze je také upravovat například MIDI zprávami v rámci hostitelské aplikace či uživatelským rozhraním hostitelské aplikace. Pokud jejich hodnotu ovlivňuje hostitelská aplikace v čase, jedná se o tzv. *automatizaci*.

Důležitou výhodou parametrů je možnost jejich uložení. Soubor všech nastavených parametrů označujeme jako *preset* a umožňuje nám rychle měnit nastavení zásuvného modulu.

2.2 Rozhraní zásuvných modulů

Existuje několik programovacích rozhraní pro komunikaci mezi hostitelskou aplikací a zvukovým zásuvným modulem. Většina z nich byla vytvořena komerčně firmou pro jejich hostitelskou aplikaci, a jejich funkčnost v ostatních softwarech je z konkurenčních důvodů omezena.

Rozhraní **VST** (Virtual Studio Technology) bylo vyvíjeno firmou Steinberg pro jejich DAW software Cubase. Toto rozhraní umožňuje vývoj pro oba majoritní operační systémy. [2]

Rozhraní **AU** (Audio Unit) bylo vytvořeno společností Apple pro software Logic Pro, který funguje pouze na systémech OS X. Funkcionalita Audio Unit je tedy omezena pouze pro tuto platformu.

AAX (Avid Audio eXtension) je speciální rozhraní firmy Avid pro pluginy určené pro využití v DAW softwaru ProTools. Lze jej pořídit jak pro operační systém OS X, tak pro systém Windows.

Alternativou pro uživatele operačního systému Linux je **LV2**. Toto rozhraní je druhou verzí původního API LADSPA (Linux Audio Developer's Simple Plugin API).

3 VST3

Rozhraní Virtual Studio Technology bylo poprvé představeno firmou Steinberg v roce 1996 v rámci softwaru Cubase 3.0. První verze byla určena pro platformu Apple Macintosh, ale o rok později byl vývoj rozšířen i pro operační systém Windows. V roce 1999 Steinberg uvedl na trh verzi 2.0, která podporovala možnosti virtuálních hudebních nástrojů. [3]

Velké změny byly představeny na počátku roku 2008 zveřejněním VST verze 3.0., která rozdělila zásuvný modul do dvou částí: **procesní část** (Processing part) a **kontrolní část** (Editing part). Procesní část má na starosti samotnou úpravu zvukového signálu. Kontrolní část spravuje grafické uživatelské rozhraní včetně uživatelských změn parametrů.

Hlavní myšlenka za tímto rozdělením bylo rozložit výkon počítače mezi jednotlivé komponenty v závislosti na jejich důležitosti. Toto naprosté oddělení dovoluje v teorii spustit procesní část na jiném zařízení než kontrolní část. [2]

Aby rozdělením nebyl narušen chod zvukového zásuvného modulu, stará se o komunikaci mezi těmito částmi hostitelská aplikace. Ta pro spuštění zvukových zásuvných modulů musí mít takového správce pluginů, který podporuje architekturu **VST Module Architecture**.[2]

3.1 Architektura VST modulů

VST Module Architecture (VST-MA) je objektově orientovaný systém, který říká hostitelské aplikaci, jak má být interpretován do ní vložený plugin. Tato sada rozhraní je nezávislá na platformě a téměř nezávislá na kompilátoru kódu. Svojí strukturou je podobná struktuře COM od firmy Microsoft.

V rámci této architektury jsou jednotlivá rozhraní reprezentována virtuálními třídami, které obsahují pouze abstraktní metody. Základním rozhraním je třída `FUnknown`, ze které jsou všechny objekty přímo či nepřímo zděděny. Tato třída se stará o to, aby každé zděděné rozhraní (třída) mělo vlastní unikátní identifikační číslo (Interface ID) a každá implementace těchto rozhraní měla vlastní identifikační číslo (Component ID).

Aby byla zajištěna kompatibilita mezi jednotlivými verzemi VST3, je využívána dědičnost pouze k „verzování“. Jakmile bylo rozhraní jednou vydáno v balíčku SDK, nesmí se již nikdy změnit. Pro potřebu vložení nových funkcí se vytvoří nové rozhraní, které dědí staré rozhraní, a tím ho rozšiřuje.

Jednotlivá rozhraní VST-MA se dělí podle „směru“ na rozhraní, která mají být implementována v hostitelské aplikaci (host imp), a na ty, které implementujeme ve zvukovém zásuvném modulu (plug imp). Některá rozhraní je možno použít oběma směry.

Aby hostitelská aplikace mohla zásuvný modul spustit, musí být plugin dodán ve formě modulu (dynamicky linkovaná knihovna pro Windows a Mach-O Bundle pro Mac platformy). Modul obsahuje jednu nebo více komponent - procesní části a jejich kontrolní části a tzv. class factory specifikující, které komponenty mají být vyvolány a jak k nim má být přistupováno. Tyto komponenty musí dědit z rozhraní `IPlugBase`, aby je mohla část hostitelské aplikace, dědící z rozhraní `IPluginFactory`, inicializovat. [2]

3.2 Procesní část

Procesní část je nejčastěji realizována odvozením z třídy `AudioEffect`. Její jádro je tvořeno děděním ze dvou rozhraní `IComponent` a `IAudioProcessor`.

Funkce třídy `IComponent` slouží k párování procesní části s kontrolní částí. Dále při inicializaci řídící části mají funkce tohoto rozhraní na starosti definici maximálního využití vstupních a výstupních sběrnic. Další funkce děděné z `IComponent` například informují o využití sběrnic.

Hlavní úlohu v procesní části má funkce `process` prostředí `IAudioProcessor`. Tato funkce má na starosti samotné zpracování signálu. Veškeré potřebné informace jsou předávány ve formě struktury `ProcessData` do argumentu funkce. Tato skupina proměnných obsahuje adresy vyrovnávacích pamětí (buffery) pro vstupní a výstupní kanály, seznam změněných parametrů, ale také informace o stavu hostitelské aplikace.

Zpracování probíhá v blocích, jejichž velikost se může s každým zavoláním této funkce změnit. Informaci o maximální velikosti bloku, módu zpracování (offline, realtime) a vzorkovací frekvenci poskytuje hostitelská aplikace formou struktury `ProcessSetup` v argumentu funkce `setupProcessing`, kterou volá před každým zavoláním funkce `process`. [2]

3.3 Kontrolní část

Kontrolní část (kontrolér) má na starosti interakci uživatele s řídící částí. Jsou zde definovány jednotlivé parametry funkce a grafické uživatelské rozhraní. Aby hostitelská aplikace rozpoznala, že tato komponenta je kontrolní částí, musí kontrolní část dědit rozhraní `IEditController`.

3.3.1 Správa parametrů

Při inicializaci kontrolní části třídy s rozhraním `IEditController` jsou také vytvářeny parametry. Parametrem je pro hostitelskou aplikaci soubor informací, mezi které patří hlavně hodnota s plovoucí desetinou čárkou v rozmezí <0,1> a k ní unikátní id. Kromě toho může být hostitelská aplikace informována o názvu, jednotkách a kroku, se kterým se má hodnota měnit.

Hostitelská aplikace se může kdykoliv kontrolní části dotázat, jak má být hodnota parametru uživateli zobrazena ve znakovém řetězci. K tomu slouží funkce `getParamStringByValue`, jejím protipólem je funkce `getParamValueByString`, která by měla vrátit normalizovanou hodnotu na základě vstupního znakového řetězce. Z těchto důvodů je výhodné si informace o chování parametru uchovávat. Ideálním prostředkem je soubor pomocných tříd, které nabízí VST3 SDK. Důležitou třídou z této skupiny je třída `EditController`, která dědí z rozhraní `IEditController`, ale obsahuje další důležité funkčnosti. Hlavní je možnost uchovávání seznamu všech parametrů a jejich interpretace. Ty si ukládá do seznamu objektů třídy `Parameter`.

Dědici třídy `Parameter` mohou specifikovat veškeré výše zmíněné požadavky a přidávat vlastní nové funkčnosti potřebné pro běh zásuvného modulu. Příkladem může být třída `RangeParameter`. Ta na základě minimálních a maximálních hodnot interpoluje z normalizované hodnoty a do normalizované hodnoty. Naopak třída `StringListParameter` slouží k přiřazení textových polí ke skokovým změnám parametru podle toho, kolik možností parametr obsahuje. Může se tedy jednat o jednoduchý dvouhodnotový přepínač nebo i komplexnější hodnotový volič.

Protože kontrolní část veškeré informace o reprezentaci parametru obsahuje, není problémem tyto informace sdílet s editačním oknem a vykreslovat uživatelské rozhraní už na základě reálných hodnot. Komplikace vznikají v procesní části, která má k dispozici pouze základní informace poskytnuté od hostitelské aplikace, a mělo by tedy být již před startem zásuvného modulu jasné, jak má být hodnota od konkrétního identifikačního čísla interpretována. Toho je docíleno definováním konstantních id pro jednotlivé parametry v procesní části.

Hodnoty parametrů se skrz hostitelskou aplikaci neposílají kontinuálně, ale pouze když dojde ke změně. Pokud dojde ke změně v kontrolní části, volá se sada funkcí `beginEdit`, `performEdit` a `endEdit` třídy `IComponentHandler`. Tato třída je implementována v hostitelské aplikaci a je prostředníkem právě pro přenos změněných parametrů. Jakmile hostitelská aplikace tyto parametrové změny přijme, zahrne je do vstupního parametru `ProcessData` procesní části vedle vedle dalších vstupních argumentů. Procesní část je tedy zpracovává ve funkci `process`.

Pokud zde dojde k další změně parametru, lze uložit do stejného datového balíku i změněné hodnoty pro výstup z této části. Ty jsou zase přes hostitelskou aplikaci poslány do kontrolní části skrz funkci `setParamNormalized`.

Pokud je plugin rozdělen svojí funkčností do několika celků (např. ekvalizér a delay), vybízejí vývojáři VST k přiřazení jednotlivých parametru k větším celkům zvaným „Units“. Units mohou být řazeny dle libosti do hierarchie, a slouží především k přehlednosti správy parametru v kontrolní části.

3.3.2 Grafické uživatelské rozhraní

Aby uživatel mohl upravit hodnoty parametrů, je běžnou praxí, že kontrolní část vyvolá i editační okno (view), které je uzpůsobeno funkci zvukového zásuvného modulu. Avšak pokud plugin nevrátí hostitelské aplikaci odkaz na rozhraní `IPlugView` při zavolání funkce kontroléru `createView`, hostitelská aplikace vytvoří editační okno vlastní. Toto základní zobrazení obsahuje běžně pouze posuvníky (slidery), a proto nemusí plně informovat uživatele o chodu pluginu.

Nastavení hodnot parametrů zásuvné aplikace ve většině případů odpovídá požadavkům na analogové zvukové zařízení, a je nutné, aby editační okno zvládalo kromě vytváření posuvných potenciometrů také otočné potenciometry (knoby nebo točítka), editovatelné textové pole a aby bylo schopno vykreslit obrázky.

Pro tyto účely byla do balíku SDK zahrnuta knihovna **VSTGUI4**, ve které jsou od SDK verze 3.6.7. k nalezení pouze dvě třídy pro vytváření editačního okna: třída `VSTGUIEditor` a třída `VST3Editor`. Obě seskupují grafické rozhraní pro kontrolu uživatelských vstupů a správu samotných uživatelských prvků – jejich umístění na obrazovku a změnu jejich hodnot.

Tyto grafické prvky musí dědit z prostředí `CView`. Jsou sdružovány do celků – kontejnerů `CViewContainer`. `CFrame` je hlavním celkem u editorů dědících z tříd VSTGUI. Do něj mohou být vloženy jak kontejnery, tak samotné uživatelské prvky. Pozice jednotlivých uživatelských prvků může být definována vytvořením nové třídy, která dědí z VST3Editor a následným přidáním jednotlivých prvků do hlavního kontejneru `CFrame`, nebo je zde k dispozici komplexní řešení s názvem *UIDescription*.

UIDescription má za úkol vytvořit skupinu uživatelských prvků, a tu pak přidat do hlavního kontejneru `CFrame`. Toho je docíleno pomocí xml souboru, ve kterém je definováno umístění a vzezření jednotlivých prvků. Zde ale funkce UIDescription nekončí. Vývojář totiž uživatelské prvky nemusí vkládat psanou definicí, ale může využít editoru uživatelských prvků. Prakticky to funguje tak, že při vytváření editoru

pomocí třídy VST3Editor je zadána cesta budoucího xml souboru. Po buildu a otevření v hostitelské aplikaci je možné kliknout pravým tlačítkem na okno zásuvného modulu, a otevřít funkci „UIDescription Editor“. Ta nabízí grafické uživatelské prostředí pro vytváření grafického uživatelského prostředí. Je zde možné s okamžitým náhledem vkládat jednotlivé uživatelské prvky, a vázat je na konkrétní parametry.

Samotné vykreslování probíhá v knihovně VSTGUI pomocí značek „špinavosti“. Při vytvoření jednoho ze základních editorů se spustí časovač `CVSTGUITimer`, který využívá interního časovače systému, a v určitých časových intervalech volá funkci editoru `notify`. Zde se aktualizuje základní `CFrame`. Aktualizace ale neprobíhá plošně – obnoveny jsou pouze ty prvky, které byly po předchozím volání časovače označeny za špinavé pomocí funkce `setDirty`. Tím pádem je obnovována pouze ta část okna, která se změnila – např. otočení knobu jednoho parametru. Další výhodou je, že tímto způsobem je zajištěna vláknová bezpečnost celého procesu.

Aby se mohla hodnota uživatelského prvku změnit, je potřeba adekvátně reagovat na uživatelské vstupy, a to především na pohyb a klik myši. K tomu slouží rozhraní `IPlatformFrameCallback`, které je děděno třídou `CFrame`. Obsahuje funkce, které jsou volány editačním oknem při každé změně pozice a tlačítkového stavu myši – tato informace je pak předána dalším prvkům které jsou obsaženy v `CFrame`.

Protože dnešní podoba zvukových efektů vychází z jejich původní historické podoby, bývá kromě zvuku často emulován i vzhled samotného přístroje. Toho bývá dosaženo několika bitmapovými obrázky, které jsou umístěny na pozadí a na jednotlivé uživatelské prvky, tak aby simulovaly vzhled reálného přístroje. Pokud přístroj má napodobit vzhled odrazivého materiálu, vzniká problém u otočných prvků, kde se prvek otáčí i s odrazem, který by se pohybovat neměl. To je pak realizováno množstvím bitmapových obrázků v jednotlivých polohách, které jsou přepínány v závislosti na hodnotě parametru. Pro toto řešení lze využít třídy `CMovieBitmap`. Pro plynulý přechod při otáčení „knobů“ je potřeba až stovky obrázků.

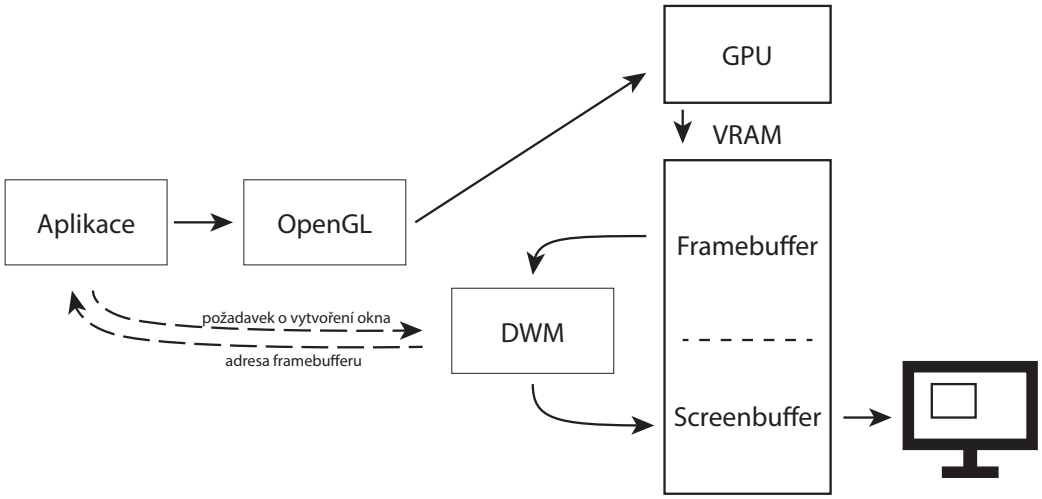
Bitmapové soubory jsou vytvářeny v externích grafických programech, kde jsou po vymodelování „vyrenderovány“ do sekvencí obrázků, což pro výslednou aplikaci ve zvukovém zásuvném modulu znemožňuje komplexnější grafické změny. S vývojem grafických karet se tento výpočet z CPU čím dál více přesouvá na rychlejší procesory GPU a výpočet je možno provádět v reálném čase. Toho využívá zejména videoherní průmysl, kde je potřeba, aby vykreslování reagovalo okamžitě na uživatelský vstup. Jádra těchto her pro komunikaci s grafickými kartami využívají grafických rozhraní (např. OpenGL či DirectX).

Protože zvukové zásuvné moduly nevykreslují pouze uživatelské prvky, ale také slouží k analýze a vizualizaci zvuku, mohlo by býti výhodné využít grafických procesorů k přenesení zatížení výkonu z centrální procesorové jednotky na jednotky grafického akcelerátoru.

4 OpenGL

Open Graphics Library (OpenGL) je multiplatformní API pro vykreslování 2D a 3D grafiky. Tato knihovna slouží pouze k vykreslování a komunikací s grafickou kartou a nezajišťuje správu oken či uživatelských vstupů. Tím OpenGL nevstupuje do specifik jednotlivých operačních systémů.

Aby se aplikace využívající OpenGL mohla vůbec zobrazit na monitoru uživatele, je potřeba ho propojit s *správcem oken* daného systému. Pro Windows od verze Vista je tímto správcem *Desktop Window Manager (DWM)* [4]. DWM má na starosti alokování části paměti grafického adaptéru (VRAM), které mohou jednotlivé aplikace využívat pro zápis výstupních informací. Těmto celkům v paměti se říká snímkové zásobníky *framebuffery*, a pokud je v zásobníku uložena celá obrazovka, hovoříme o něm jako o obrazovém zásobníku – *screenbufferu*. DWM následně tyto framebuffery jednotlivých oken sloučí, a na základě informací o překrývání z nich vytvoří výsledný obraz, který je poslán na výstup grafického adaptéru. Viz obr. 1.



Obrázek 1: Tok dat při vykreslování grafické aplikace.

4.1 Rozdíly ve verzích OpenGL

Protože rozhraní OpenGL bylo poprvé zveřejněno v roce 1992, a nároky na grafické karty se od té doby jen zvyšovaly, musela struktura OpenGL být s tímto vývojem pravidelně upravována. Verze **1.0** obsahovala základní funkce pro práci s body, barvami a texturami. Také zde byl představen základní model osvětlovacího systému a maticových počtů. [5]

V dalších úpravách až do verze 2.0 se rozšiřovaly možnosti verze 1.0 o např. 3D textury, komprimované textury, automatickou generaci mipmap, upravené ukládání

datových polí do rychlé vyrovnávací paměti grafického akcelerátoru a mnohé další úpravy, které usnadňují či zrychlují práci s původní verzí OpenGL.

Aby vývojáři měli větší kontrolu nad fungováním grafického adaptéru, byla hlavním přídavkem do knihovny OpenGL verze **2.0** (2004) podpora *shaderů* (viz kapitola 4.3). [6]

Ve verzi **3.0** (2008) byl, kromě mnoha dalších změn, představen nový přístup k číslům s plovoucí desetinou čárkou a možnost vykreslovat místo do výstupních framebufferů také do textur a pomocných vyrovnávacích pamětí. Verze 3.0 kromě uvedení nových příkazů, označila příkazy staré, které by neměly být dále používány. Všechny tyto zastaralé funkce byly následně ve verzi **3.1** (2009) odstraněny. [6]

OpenGL bylo poté dále vyvíjeno přes několik verzí až do verze **4.5** (2014). Za tu dobu přibylo mnoho pokročilých funkcí pro novější grafické karty. Na začátku roku 2016 bylo zveřejněno rozhraní **Vulcan**, které má za úkol sjednotit klasické OpenGL s odnoží OpenGL pro mobilní zařízení **OpenGL ES**. Kromě toho má zefektivnit propojení CPU a GPU. [7]

4.2 Doplňující knihovny

Aby grafický program mohl zakreslovat grafický obsah do okna, je nutné navázat se správcem oken a OpenGL spojení. Vývojář může toto propojení naprogramovat sám, běžněji se ale používají tzv. *windowing library* - knihovny, jejichž funkce mohou požádat DWM o vytvoření okna, specifikují jeho velikost, název nebo ikonu. DWM na tuto žádost odpoví adresou na framebuffer pro dané okno. Do tohoto framebufferu už může OpenGL s využitím svých vlastních funkcí vykreslovat. Mezi nejznámější knihovny, které dovolují vykreslovat OpenGL, patřila donedávna knihovna **OpenGL Utility Toolkit (GLUT)**, ale jehož použití se dnes již nedoporučuje kvůli její nepodpoře vyšších verzí OpenGL[9]. Alternativou jsou knihovny **freeGLUT** [11] či **GLFW** [12]. V případě zvukových zásuvných modulů se o toto spojení stará hostitelská aplikace.

Funkce OpenGL se od verze 1.1 nevolají přímo, ale pomocí funkcí tzv. *loading library* [10]. Je tím zajištěna podpora pro více platforem. Mezi používané loading library patří: **GLEW**, **GL3W**, **glLoadGen**, **glad** a další.

Pro tvorbu 3D aplikací se hodí využít i *matematické knihovny*. Jejich funkce dovolují počítat s vektory a maticemi. Lze například přenést 3D data do perspektivního zobrazení. Nejznámější knihovnou je **glm**. Ta je navržena tak, aby odpovídala matematickým funkcím v *OpenGL Shading Language*. Viz kapitola 4.3.1.

4.3 Shadery

Kvůli náročnosti grafických výpočtů probíhá tato činnost uvnitř grafického adaptéru počítače v procesorech GPU. Využívá se toho, že většina zpracovávaných dat je na sobě nezávislá a lze s nimi pracovat paralelně. Pro toto velké množství procesorů je

potřeba psát programy jinak než pro aplikaci, jejíž výpočet řídí CPU. Program, který je určen pro grafické procesory, se nazývá *shader* a je psán v jazyce příslušejícím dané knihovně, která do paměti grafické karty zapisuje.

4.3.1 Jazyk GLSL

Jazyk odpovídající knihovně OpenGL se jmenuje *OpenGL Shading Language (GLSL)*. Vychází ze syntaxe jazyka C, ale má několik zásadních rozdílů. GLSL využívá svoji vlastní základní knihovnu, ve které jsou k nalezení specifické funkce, datové typy, rozhraní a systémem předdefinované proměnné.

GLSL definuje *shader* jako zkompilovaný kus kódu určený pro jednu část výpočtu grafického řetězce. *Grafický program* je označení pro kaskádu shaderů – řetězec výpočtů, jehož výstupem je výsledný obraz.

Vstupní (*ins*) a výstupní (*outs*) proměnné jednotlivých fází grafického řetězce jsou definované funkcemi jednotlivých shaderů. Protože jsou shadery řazeny za sebe, je nutné, aby první článek řetězce převzal všechny potřebné proměnné, a i když je sám nepotřebuje, aby je poslal dál tam, kde potřeba jsou.

Vývojář může využívat i uživatelských proměnných, které nejsou vázány na funkci shaderu. Těmto proměnným se říká *uniforms*. Jejich hodnota ale nemůže být měněna v průběhu zpracovávání programu – chovají se jako konstanty.

Před užitím programu je potřeba do VRAM uložit jednotlivé vstupní a uživatelské proměnné. To je prováděno pomocí OpenGL příkazů v OpenGL kontextu aplikace. Tyto proměnné jsou pak globálně deklarovány na začátku každého kódu shaderu.

Zpracování proměnných probíhá uvnitř funkce `main` podobně jak v jazyce C. Do výstupních proměnných se uloží výsledná hodnota, kterou pak využije další článek řetězce. Prvním článkem řetězce musí být vždy vertex shader a koncovým článkem smí být pouze fragment shader. [8]

4.3.2 Funkce shaderů

Uživatelsky programovatelné shadery se dělí podle své funkce na *vertex shadery*, *geometry shadery*, *fragment shadery* a *compute shadery*.

- **Vertex shader** má na starosti úpravu vrcholů - tzv. vertexů. Shader může změnit pozici, barvu či uv souřadnice vertexu. Každému bodu je přiřazen jeden vertex shader, takže nelze provádět změny s globálnějším účinkem.
- **Geometry shader** – Tento shader zpracovává jednoduchý objekt (čára, trojúhelník) a vytváří z něj objekty nové nebo upravuje stávající.
- **Fragment shader** zpracovává barvu a hloubku bodu, který se v budoucí době může stát pixelem. Dopočítává se zde například jeho barva v oblasti mezi třemi vertexy.

- **Compute shader** – Tento shader se v OpenGL objevil až ve verzi 4.3 a slouží k dalším výpočtům, které je lépe provádět paralelně.

5 Implementace OpenGL do VST3

Protože VST3 a OpenGL jsou povahově naprosto odlišné knihovny, jejich skloubení provází mnoho obtíží. Existuje proto limitované množství možností pro implementaci OpenGL do VST3.

5.1 Qt

Qt je multiplatformní C++ framework, který svojí strukturou dovoluje psát interaktivní aplikace. Knihovny tohoto aplikačního rámce si zakládají na několika způsobech vytváření uživatelského prostředí, ale obsahují také pokročilé funkce pro zpracování zvuku, obrazu a dalších dat. Tento framework je dodáván s vlastním správcem projektů, textovým editorem a editorem uživatelského prostředí, ale je možno využít i externích nástrojů.

Tvorba uživatelského prostředí využívá modulárního systému *Widgets*, který se stará také o vytváření okna. Jedna z pomocných tříd umožňující vytvořit OpenGL kontext se jmenuje `QOpenGLWidget`. Veškeré doplňující, loadovací a matematické knihovny jsou přítomny v knihovnách Qt. Qt zvládá i nahrávání a práci s shadery a texturami.[13]

Samotná implementace do VST pluginu by probíhala vytvořením objektu, která by dědila z třídy `QOpenGLWidget`, v kontrolní části pluginu. Zde by bylo potřeba zajistit přetypování na rozhraní `IPlugView`, které vyžaduje hostitelská aplikace.

5.2 JUCE

Multiplatformní framework JUCE slouží k vytváření interaktivních audiovizuálních aplikací. Od frameworku Qt se liší důrazem na tvorbu audio efektů. Jeho součástí je knihovna funkcí **JUCE 4.1** a aplikace pro správu projektů **PROJUCER**.

Při tvorbě zvukových efektů umožňuje PROJUCER s jedním zdrojovým kódem vytvořit efekt pro více API (VST, AU i AAX) pomocí tzv. „wrapperů“. Vývojář tedy píše kód pouze s využitím tříd knihovny JUCE, které v závislosti na platformě a rozhraní zavolají odpovídající funkci.

PROJUCER má také zabudovaný editor zdrojového kódu, který je kompilován kontinuálně. Dále obsahuje editor uživatelského rozhraní a generátor projektů pro většinu vývojových prostředí (IDE). I když se JUCE orientuje převážně na tvorbu zvukových zásuvných modulů, je možné pomocí této sady nástrojů vytvářet i samostatné aplikace (*standalone*) – programy běžící bez hostitelské aplikace.

Knihovna JUCE obsahuje množství pomocných tříd pro vytváření především

zvukových, ale také grafických aplikací. Jsou zde k nalezení knihovny pro správu vstupů a výstupů, správu oken a kontrolu zpráv MIDI.

Pro implementaci OpenGL je možno využít vestavěných tříd `OpenGLContext` a `OpenGLRenderer`. Kontext je vytvořen v `OpenGLContext` a samotné funkce jsou k vykreslení volány ve funkci `renderOpenGL` objektu `OpenGLRenderer`. Knihovna JUCE nabízí několik pomocných funkcí a tříd, které při psaní OpenGL příkazů mohou usnadnit práci. Příkladem je třída `OpenGLHelpers`. [14]

Pro implementaci pak stačí vložit objekt dědící z `OpenGLRenderer` do editoru efektu jako další uživatelský prvek funkcí `addAndMakeVisible`.

5.3 VSTGUI4

Od verze VSTGUI 4.1 je zde možné vytvořit i třídu `COpenGLView`. Tato třída má zabudované funkce pro tvorbu OpenGL kontextu a pro základní vykreslování na platformách Windows a Mac Cocoa. `COpenGLView` dědí ze třídy `CView` stejně jako ostatní ovládací prvky, které VSTGUI poskytuje a lze ji tedy jednoduše přidat do editačního okna. Tato skutečnost také dovoluje využít funkce, které jsou zavolány při aktivitě myši v oblasti, kde je vykreslováno OpenGL.

Základní nastavení je vhodné provádět ve funkci `platformOpenGLViewCreated`. Může se tak jednat například o kompilaci a linkování shaderů, definování transportních bufferů, nastavení čistící barvy nebo inicializaci podpůrných knihoven. Protože `COpenGLView` je stavěné na OpenGL verzi 1, je právě nutné pro využití plné funkčnosti této vykreslovací knihovny využít loadovací knihovny. Obdobně slouží funkce `platformOpenGLViewWillDestroy`, která by měla být využita pro smazání shaderů a bufferů před zavřením okna.

Samotné vykreslování probíhá ve funkci `drawOpenGL` vždy, když je tato třída označená za špinavou. Aby se mohlo přistupovat k OpenGL kontextu, je potřeba kontext uvnitř této funkce navázat funkcemi `makeContextCurrent`, `lockContext` a `unlockContext`. Mezi uzamknutím a odemknutím je už možno volat funkce OpenGL, které zapisují informace do *dvojitého bufferu*. Po vykreslení lze k záměně vyrovnávacích pamětí využít funkce `swapBuffers` .

Bohužel v dnešní fázi je knihovna VSTGUI ještě nedokončená a některé funkce nejsou vůbec implementovány. Jedná se tak například o nastavení antialiasingu při vytváření kontextu pomocí třídy `PixelFormat`, která má být předána při vytváření platformy pro využití funkce dvojitého bufferu, zadání bitové hloubky nebo multisamplingu. To ve svém důsledku zabraňuje úmyslu vytvořit lepší estetický vjem na koncového uživatele.

6 Virtuální mikrofony

V praktické části této bakalářské práce je realizován tzv. *virtuální mikrofon*. Než však bude v této kapitole vysvětleno, co je myšleno pod tímto pojmem, je nutné popsat směrové vlastnosti mikrofonů obecně a jejich vizualizaci.

Směrová charakteristika je vlastnost popisující citlivost mikrofonu v závislosti na směru přicházejícího zvukového signálu. Je dána konstrukcí komory mikrofonu a důležitým prvkem je zde přístup vzduchu k membráně.

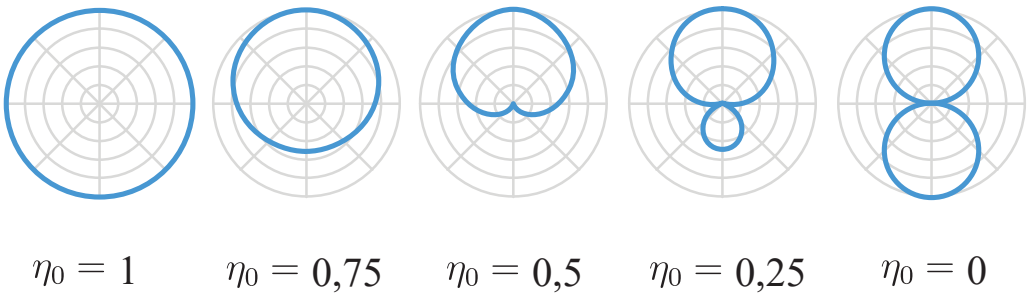
Pokud je přístup akustické vlny k membráně z obou jejích stran, hovoříme tak o mikrofonu *I. řádu*, jehož hlavní citlivostní směr se nachází kolmo na plochu membrány. Pohyb membrány tedy reprezentuje rozdíl akustického tlaku jedné a druhé strany. Pokud tedy zvuková vlna dorazí na membránu kolmo na její osu, je tlak na obou stranách membrány stejný, a membrána se nerozpohybuje.

Pokud je jedna ze stran membrány plně zakryta, jedná se naopak o mikrofon *0. řádu*, jehož membrána reaguje na veškeré tlakové změny. Zde je již nepodstatné, odkud zvukový signál přišel, a označujeme jej tak jako *všesměrový*.

Pokud membránu z jedné ze stran zakryjeme pouze částečně, docílíme tak kombinovaného mikrofonů nultého a prvního řádu. Míra zakrytí pak určuje tvar směrové charakteristiky, a tento tvar můžeme znázornit v *kruhovém diagramu*. Kruhový diagram graficky znázorňuje citlivost mikrofonu η v závislosti na směru φ a míře zakrytí η_0, která odpovídá normalizované citlivosti mikrofonu 0. řádu. Citlivost kombinovaného mikrofonu pro potřeby vykreslení do kruhového diagramu pak znázorňuje rovnice 1

$$\eta = |\eta_0 + (1 - \eta_0) \cos \varphi| \tag{1}$$

Na základě tvarů jsou pak i jednotlivé mikrofony pojmenovány. Mikrofon nultého řádu se taktéž označuje jako kulový, prvního řádu jako mikrofon s omičkovou charakteristikou, a u případu, kde míra zakrytí je pouze poloviční, mluvíme o kardioidní (ledvinové) charakteristice.



Obrázek 2: Interpolace mezi mikrofonem 0. řádu až po 1. řád.

Této kombinace je možno docílit již zmíněným zakrytím jedné strany membrány. Další možností pro vytvoření kombinovaného mikrofonu je použití vícero mikrofonů

nultého nebo vyššího řádu, a následné váhované spojení jejich signálů. Lze tak například vytvořit mikrofon prvního řádu pomocí dvou mikrofonů nultého řádu blízko sebe, když jejich signály od sebe odečteme.

Této kombinaci a tomuto procesu říkáme *tvarování směrové charakteristiky* a stejný princip je uplatňován i v *ambisonii* – technice snažící se zabrat veškerý okolní zvuk pro potřeby vytvoření vícekanálového prostorového zvuku. Toho je docíleno zpravidla pomocí jednoho všesměrového mikrofonu W, a tří „osmičkových" mikrofonů XYZ natočených kolmo na sebe. Této mikrofonové sestavě se říká *B-Format*.

Kombinací signálů těchto čtyř mikrofonů je pak možné „dekódovat" a získat signál z jakéhokoliv směru a s jakoukoliv směrovou charakteristikou. Tím vznikne nový pomyslný mikrofon, tzv. *virtuální mikrofon*. Parametry pro výpočet výsledného signálu jsou azimut φ a elevace ϑ. Převodem sférických souřadnic na kartézské lze dosáhnout hodnot jednotlivých zesílení pro mikrofony XYZ (2).

$$\begin{aligned} g_x(\varphi, \vartheta) &= \cos\varphi \cos\vartheta, \\ g_y(\varphi, \vartheta) &= \sin\varphi \cos\vartheta, \\ g_z(\varphi, \vartheta) &= \sin\vartheta. \end{aligned} \tag{2}$$

Po aplikování těchto zesílení na vstupní signály, a následném sečtení těchto signálů dostáváme virtuální mikrofon 1. řádu natočený kýženým směrem. Pro změnu směrové charakteristiky je potřeba tento virtuální mikrofon nakombinovat s mikrofonem nultého řádu. To i předchozí krok popisuje následující rovnice 3

$$s(t, \varphi, \vartheta) = g_w s_w(t) + (1 - g_w)[g_x(\varphi, \vartheta)s_x(t) + g_y(\varphi, \vartheta)s_y(t) + g_z(\vartheta)s_z(t)], \tag{3}$$

kde $s(t, \varphi, \vartheta)$ je výstupní signál virtuálního mikrofonu, $s_w(t), s_x(t), s_y(t), s_z(t)$ jsou vstupní signály jednotlivých mikrofonů a $g_w, g_x(\varphi, \vartheta), g_y(\varphi, \vartheta)$ a $g_z(\vartheta)$ jsou váhovací činitele (zesílení) jednotlivých mikrofonů.

Pro ambisonii lze zvolit i další sestavu mikrofonů, jež je tvořena čtyřmi stejnými mikrofony subkardioidní směrové charakteristiky. Tedy kombinovaného mikrofonu, kde je podíl složky mikrofonu nultého řádu ke složce prvního řádu 7/3. Tyto akustické přijímače jsou rozestaveny do kruhu s hodnotami azimutu a elevace $(\pi/4, \pi/5), (3\pi/4, -\pi/5), (-\pi/4, -\pi/5)$ a $(-3\pi/4, \pi/5)$. Tyto mikrofony jsou v rámci této práce označeny jako LF, RF, LB, RB v závislosti na jejich směru. Tato mikrofonová konfigurace se označuje jako *A-Format*. [16]

Činitelé zisku závislé na úhlech elevace a azimutu pro jednotlivé vstupní signály jsou popsány v rovnicích 4 a jsou odvozeny z [16]

$$\begin{aligned}
g_{LB}(\varphi,\vartheta) &= \frac{g_w + (1-g_w)[-g_x(\varphi,\vartheta) + g_y(\varphi,\vartheta) - g_z(\vartheta)]}{2},\\
g_{LF}(\varphi,\vartheta) &= \frac{g_w + (1-g_w)[g_x(\varphi,\vartheta) + g_y(\varphi,\vartheta) + g_z(\vartheta)]}{2},\\
g_{RF}(\varphi,\vartheta) &= \frac{g_w + (1-g_w)[g_x(\varphi,\vartheta) - g_y(\varphi,\vartheta) - g_z(\vartheta)]}{2},\\
g_{RB}(\varphi,\vartheta) &= \frac{g_w + (1-g_w)[-g_x(\varphi,\vartheta) - g_y(\varphi,\vartheta) + g_z(\vartheta)]}{2}.
\end{aligned} \tag{4}$$

Výsledný signál virtuálního mikrofonu využívající vstupních signálů systému A-Format lze pak vyjádřit rovnicí 5

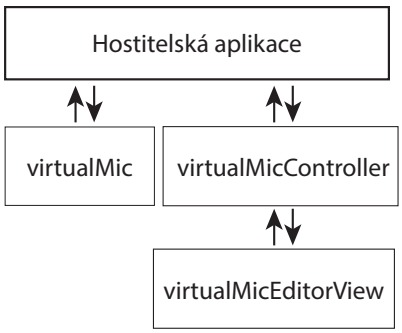
$$\begin{aligned}
s(t,\varphi,\vartheta) = {} & g_{LB}(\varphi,\vartheta)s_{LB}(t) + g_{LF}(\varphi,\vartheta)s_{LF}(t)\\
& + g_{RF}(\varphi,\vartheta)s_{RF}(t) + g_{RB}(\varphi,\vartheta)s_{RB}(t).
\end{aligned} \tag{5}$$

Ambisonie umožňuje pořizovatelům zvukového záznamu značnou flexibilitu ve tvorbě prostorového zvuku, kde je volba zaznamenávaného směru přesunuta do průběhu postprodukce. Tímto způsobem je také možné vytvořit více signálů než bylo původních mikrofonů. Technika se však hodí spíše pro zachycení okolních ruchů než pro přesný záznam např. mluveného slova. Je zde totiž omezená „čitelnost“ směru, ze kterého by zvuk měl přicházet při prostorovém poslechu. Z tohoto důvodu se pro kombinaci využívá i více mikrofonů, aby konkrétnost výsledného virtuálního mikrofonu byla co největší – tzv. ambisonie vyšších řádů.

7 Zásuvný modul VirtualMic

Zvukový zásuvný modul virtualMic slouží k syntéze čtyř zvukových signálů v jeden až osm výstupních signálů s volitelnými parametry pro směrovost a natočení až osmi virtuálních mikrofonů. Vstupními signály se předpokládá záznam ze sady mikrofonů v konfiguraci A-Format nebo B-Format. Zásuvný modul je také opatřen grafickým uživatelským rozhraním, jehož prvky využívají technologii OpenGL.

Modul je strukturován standardně podle architektury VST3, tedy na procesní a kontrolní část. Třída `virtualMic` zastává funkci procesní části, a komunikuje s kontrolní částí `virtualMicController` pouze prostřednictvím hostitelské aplikace. Kontrolní část vyvolává a spravuje editační okno `virtualMicEditView`, které zobrazuje na obrazovku jednotlivé grafické prvky. Grafické prvky kombinují 3D grafiku s vektorovou grafikou, proto byly v rámci této práce vytvořeny i dvě provázané pomocné knihovny **VST3D** a **VSTGLUI**.



Obrázek 3: Komunikační struktura zásuvného modulu virtualMic.

7.1 Procesní část

Do procesní části by měly vstupovat aspoň 4 vstupy a vystupovat nejlépe 8 výstupů, a tuto informaci je potřeba sdělit hostitelské aplikaci, aby mohla přizpůsobit své prostředí tomuto faktu. Učiněno je tak ve funkci `setBusArrangements`, která je volána hned po inicializaci procesní části, a také před každým voláním funkce `setupProcessing`. `SetBusArrangements` reaguje na konfiguraci vstupů a výstupů hostitelské aplikace a vrací informaci, zda-li zásuvný modul může v této konfiguraci pracovat. Může tedy proběhnout několik volání této funkce, než se dosáhne kladného výsledku. Tato část virutalMic pluginu reaguje kladně na jakékoliv množství výstupních sběrnic, zato vstupní sběrnice musí být alespoň čtyři. Tyto vstupy a výstupy jsou náležitě pojmenovány, pokud zásuvný plugin zaregistruje konkrétní reproduktorovou konfiguraci A-Format nebo B-Format.

Po nastavení sběrnic je volána funkce `setState`, která slouží k načítání uložených hodnot v případě, že byl zásuvný plugin již dříve spuštěn nebo když bylo zvoleno přednastavené nastavení pluginu hostitelské aplikace (preset). Pomocí streamu `state` je možno nahrát jednotlivé proměnné, které byly v minulosti uloženy pomocí funkce `getState`. Jednotlivé proměnné jsou ukládány do lokálních proměnných třídy `virtualMic`. Načítání a ukládání zde probíhá pouze formálně, protože aktuální hodnoty parametrů jsou ještě před zpracováním zvukového signálu poslány z kontrolní části.

Funkční struktura zásuvného modulu probíhá v *jednotkách virtuálního mikrofonu*. Tyto jednotky jsou v tomto případě pouze čtyři, přičemž každá z nich může být klonována zrcadlově symetricky v ose X. To dohromady dává možnost osmi virtuálních mikrofonů, ale pouze s čtyřmi možnými parametrovými nastaveními, protože každý mikrofonový zrcadlový pár nastavení parametrů sdílí. Každá jednotka je přisouzena ke dvěma zvukovým výstupům.

Informace o tom, zda-li je jednotka virtuálního mikrofonu vypnuta, zapnuta nebo v zrcadlovém režimu, je uložena v poli proměnné `iMicMode`. Hodnota 0 značí neaktivitu dané jednotky, hodnota 1 funkci v mono módu, kdy je do obou výstupních cest posílán stejný signál, hodnota 2 určuje režim zrcadlového nastavení. V tomto případě je do výstupu posílán signál v závislosti na natočení jednotlivých virtuálních mikrofonů.

Pro každou jednotku je zde uložena taktéž informace o natočení v rovině XZ pod názvem `fAzimuth` a elevace od této plochy s názvem `fElevation`. Obě tato pole veličiny pracují v jednotkách radiánů. Pro každou mikrofonní jednotku jsou zde uloženy i činitele směrovosti mikrofonu `fDirectivity` a zesílení jednotky `fUnitGain`. V případě směrovosti odpovídá hodnota 0 směrovosti mikrofonu prvního řádu, zato hodnota 1 značí všesměrový mikrofon nultého řádu. Hodnoty mezi těmito limity jsou poměrovou kombinací těchto dvou charakteristik, kdy v bodě 0.5 se nalézá běžně známá kardioidní charakteristika. Hodnoty zesílení se taktéž nalézají v tomto číselném rozmezí, avšak jejich interpretace je klasická pro digitální zpracování signálu, a to 0 zesílení $-\infty$ dB a 1 zesílení 0 dB.

Tyto parametry je pak možné nalézt i v kontrolní části, kde je například hodnota úhlů uložena v stupních pro lepší orientaci koncového uživatele. Co však v kontrolní části není, jsou hodnoty zesílení pro jednotlivé vstupy pro každý virtuální mikrofon. Tyto hodnoty jsou uloženy v matici `gA` pro konfiguraci A-Format a `gB` pro konfiguraci B-Format. Ze vzorce 4 je jasné, že hodnoty pro zesílení v konfiguraci A-Format se počítají z hodnot v konfiguraci B-Format, a proto je vhodné ukládat obě tyto hodnoty, i když celý systém může běžet pouze v jedné konfiguraci.

Hodnoty těchto zesílení se vypočítávají v interní funkci `ComputeGains` podle vzorců 2 a 4. Od této funkce se očekává, že bude volána poměrně často, a proto upravuje vždy pouze část matice, která se týká jen jedné mikrofonní jednotky.

Největší práce ale probíhá ve funkci `process` této procesní části. Funkce `process` by měla být volána ze speciálního výpočtového vlákna hostitelské aplikace a zastává dvojí funkci zpracování změněných parametrů a úpravu zvukového signálu.

Změněné parametry jsou sem směřovány skrz strukturu procesních dat – konkrétně přes rozhraní `IParameterChanges`. Funkce data projde a lokální proměnné aktualizuje na základě hodnot změněných parametrů a zároveň zapíše, k jaké mikrofonní jednotce data patří na základě předem definovaných identifikačních čísel. Tato čísla jsou definována v souboru `virtualMicParams.h`, kde jsou ve formě výčtu uvedeny všechny použité parametry.

Pro každou mikrofonní jednotku je pak zavolána již dříve zmiňovaná funkce `ComputeGains` tak, aby i hodnoty matic `gA` a `gB` byly platné. Je nutno podotknout, že parametry nejsou již dále měněny v průběhu zpracovávání signálu a je zaznamenán vždy pouze poslední stav při změně parametru. Prostředí VST samozřejmě dovoluje možnost kontinuální změny v průběhu, obzvláště při automatizaci, nicméně dokud vyrovnávací paměť nepřesáhne příliš znatelné velikosti, tak aby zvukový signál nebyl zpracováván téměř v reálném čase, je skoková změna parametru každou zpracovávanou várkou nepostřehnutelná.

Nejdůležitější část zvukového pluginu se nachází pod sekcí pro ukládání změněných parametrů, a to je zpracování zvukového signálu. Zde je na začátku ze struktury procesních dat data vyseparován ukazatel na vstupní a výstupní vyrovnávací paměti. Pak je cyklem `for` procházen každý výstup – tedy každý virtuální mikrofon. V každé této smyčce je zjištěno, zda-li je jednotka, ke které mikrofon náleží, zapnutá. V kladném případě je započata další smyčka, která pro

každý ze čtyř vstupů provede propočet, jakou měrou by se měl podílet na výstupním signálu. Je zde taky zohledňována konfigurace vstupních dat. S A-Format konfigurací je výstupní zisk vstupu roven obsahu matice pro daný příchozí signál vypočítaný již ve funkci `ComputeGains` podle rovnice 5. U B-Format konfigurace je zisk definován rovnicí 3 a algoritmicky aplikován dle následujících pravidel:

Zdrojový kód 1: Určení zisku pro jeden z mikrofonů B-Format.

```
if (i == 0) { // W Mic
  gain = gB[m][0];
} else { // X,Y or Z Mic
  gain = (1 - gB[m][0]) * gB[m][i];
}
```

kde m je číslo virtuálního mikrofonu (0-7) a druhé číslo matice představuje pořadí vstupů (0-3).

Poté je každý vzorek vstupního signálu násoben zesilovacím činitelem `gain`, celkovým ziskem dané mikrofonní jednotky, a hned následně přičten do výstupní vyrovnávací paměti daného virtuálního mikrofonu.

Po skončení obou v sobě vnořených smyček se vrátí hostitelské aplikaci hodnota o úspěšném provedení procesního volání `kResultOk`.

7.2 Kontrolní část

Aby procesní část mohla bez problémů pracovat, je potřeba jí poskytovat dostatečné informace o parametrech. Této úlohy se zhošťuje třída `virtualMicController`, jež je vyvolána při spuštění zásuvného modulu ihned poté, co se vytvoří procesní část. Po úspěšné inicializaci proběhne ze strany hostitelské aplikace několik dotazů na informace o parametrech, které již třída `virtualMicController` ve funkci `initialize` vytvořila. Proběhlo zde vytvoření celkem 21 parametrů, 5 pro každou mikrofonní jednotku a jeden „globální“ parametr vstupní konfigurace.

Pomocí pomocných parametrových tříd `RotationParameter` a z ní odvozené třídy `HalfRotationParameter`, definovaných v souboru `virtualMicParams`, se vytvořily objekty obsahující nezbytné informace pro určení parametru Elevace a Azimutu virtuálního mikrofonu. Tyto pomocné třídy pak pomáhají všem částem zvukového modulu pomocí statických funkcí interpretovat normalizovanou hodnotu jak do radiánů, tak do stupňů.

Dalšími objekty parametrů určující činitel směrovosti „Directivity index“ a parametr určující zisk mikrofonní jednotky jsou třídy `FloatParameter` a `GainParameter`, které pouze rozšiřují základní třídu `Parameter` o konkrétnější určení výstupního formátu znakového řetězce. Prakticky to znamená, že u parametru zisku je normalizovaná hodnota přepočtena na jednotky decibel.

Parametr pro určení módu mikrofonní jednotky a pro určení vstupní signálové konfigurace jsou definovány pomocnou třídou `StringListParameter`. Pro ni je vytvořen seznam znakových řetězců – možných hodnot, kterých mohou parametry nabývat.

Pro budoucí vývoj zásuvného modulu jsou zde definovány i další parametry, ty však vytvořeny nejsou, jelikož jsou zapouzdřeny v podmínce, která se táže na verzi zásuvného modulu. Každý z parametrů je označen identifikačním číslem dané jednotky (unit), tak aby bylo možné později zjistit, ke které mikrofonní jednotce parametr patří. Bohužel funkčnost tohoto systému se zdá býti omezená faktem, že identifikační čísla jednotlivých parametrů mají, v tomto konkrétním případě, v sobě mezery, a tedy třídy VST SDK špatně přiřadí číslo jednotky. To je v celém zásuvném modulu řešeno několika podmínkami tážící se na rozmezí, ve kterém se identifikační čísla pro danou jednotku nachází.

Po inicializaci kontrolní části je již možné na sérii dotazů o parametrech hostitelské aplikaci odpovědět, to však řeší vnitřní funkce třídy `EditController`, ze kterých třída `virtualMicController` dědí. Jakmile hostitelská aplikace má informaci o všech vytvořených parametrech, zavolá funkci `createView`, která slouží k vytvoření editačního okna. Editační okno je v případě zásuvného modulu určeno třídou `virtualMicEditorView` (viz další kapitola 7.3)

V případě, že hostitelská aplikace má uložené hodnoty o stavu zásuvného modulu z dřívějšího spuštění nebo díky přednastaveným nastavením, volá funkci `setState`, která všem parametrům tyto hodnoty přisoudí. Činí tak cyklicky s využitím funkce `loadParam`, která zpracovává stream z hostitelské aplikace. Obdobně pracuje funkce `getState` a `saveParam`.

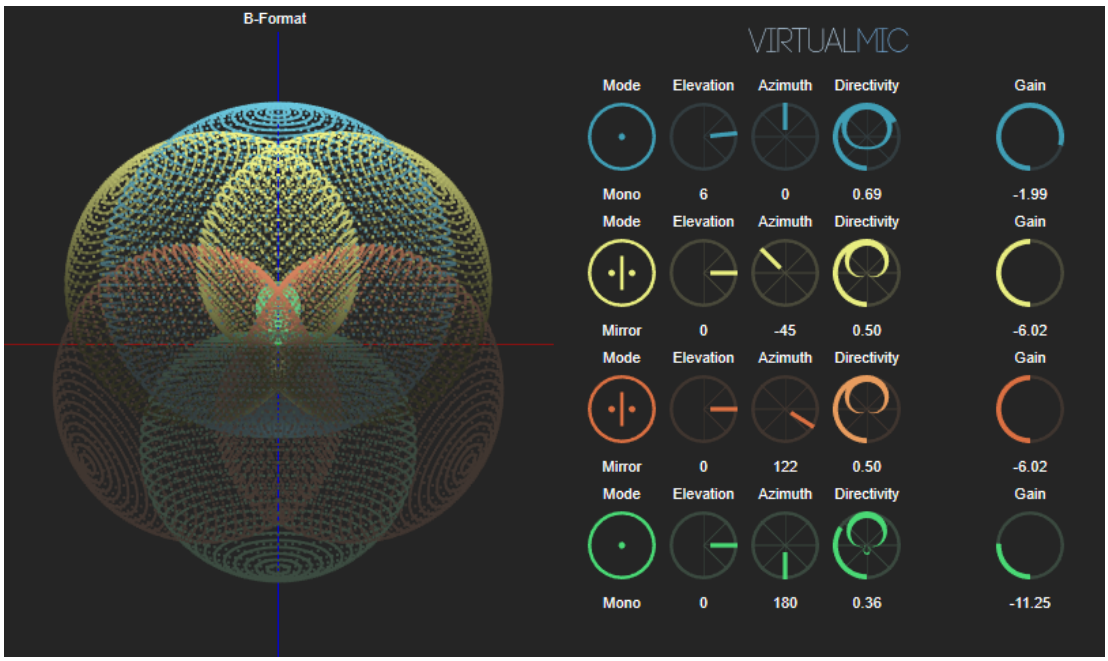
V kontrolní části je taktéž definován postup, pokud je nějaký z parametrů změněn. To je provedeno ve funkci `setParamNormalized`, jež je volána nejen z útrob kontrolní části, ale také z hostitelské aplikace. V této situaci se jednak změní hodnota parametru, následně se odešle editačnímu oknu funkce `update` s hodnotou identifikačního čísla parametru a novou hodnotou. Tímto způsobem je možné udržovat editační okno vždy aktuální.

7.3 Editační okno

Editační okno vznikne krátce po spuštění kontrolní části, pokud hostitelská aplikace požádala o jeho otevření. Může se tak pravidelně stávat, že je zásuvný modul v provozu – má spuštěnu procesní a kontrolní část, ale editační okno je zavřené.

Editační okno pluginu VirtualMic lze rozdělit na dvě části; vlevo vizualizační okno, ve kterém probíhá náhled směrových charakteristik v prostoru, vpravo interaktivní uživatelské prvky, které mění hodnotu parametrů.

Vizuální podoba editačního okna je vytvářena funkcí `open` a zavírána funkcí `close`. Ve funkci `open` probíhá vše důležité k definování vizuálního stylu zásuvného modulu. V první fázi je zkontrolováno, zda-li se tato funkce nevolá podruhé a není již editační okno otevřeno. Pokud otevřeno není, vytvoří se hlavní konstrukční prvek editačních oken využívající VSTGUI a to `CFrame`, do kterého budou ukládány veškeré uživatelské prvky a kontejnery uživatelských prvků. Je zde definována barva pozadí a následně je sem vložen první prvek – logo zásuvného modulu. Jedná se o jednoduchý obrázek, jehož vykreslení zaštiťuje knihovna VSTGUI.



Obrázek 4: Vzhled editačního okna zásuvného modulu VirtualMic.

Následně je přidána čtveřice uživatelských kontejnerů `VirtualMicContainer`, která obsahuje všechny uživatelské prvky pro jednu mikrofonní jednotku. Vzezření `VirtualMicContainer` je definováno v souboru `virtualMicGUI`. Pro zjednodušení vytváření prvků je každý z parametrů zastoupen třídou KnobText, která sdružuje dohromady vždy textový popisek názvu parametru (`CTextLabel`), textové pole pro zadání konkrétní hodnoty (`CTextEdit`) a grafický knob, jehož otáčením se docílí změny parametru. Pokud je parametr děděn ze třídy `StringListParameter`, je k němu vytvořena třída `KnobMenu`.

Zde je využito kombinované techniky vykreslování, kdy textová pole jsou vytvářena s pomocí klasického VSTGUI přístupu, naopak pro vytvoření virtuálního otočného potenciometru byla vytvořena speciální knihovna využívající sice třídy `COpenGLView` rozhraní VSTGUI, ale vykreslování probíhá s využitím vyšších verzí OpenGL. Tato knihovna byla pojmenována **VSTGLUI** a jedná se, ve své podstatě, pouze o soubor několika točítek s velmi specifickou funkcí pro potřeby zásuvného modulu VirtualMic (viz kapitola 7.5).

Každé mikrofonní jednotce – `VirtualMicContaineru` je přiřazena trojice barev, která určuje barevnosti všech prvků v tomto kontejneru. Jedná se o primární a sekundární barvu, a také o barvu pro situaci, kdy se kurzor myši nachází nad uživatelským prvkem.

`VirtualMicContainer` také obsahuje jeden jediný kontejner – součást VSTGLUI, který sdružuje všechny grafické prvky, které jsou použity z této knihovny. Proměnná, která tento další kontejner obsahuje, se jmenuje `openglContainer`. Knihovna VSTGLUI v tomto nabízí jisté omezení, protože byla vytvářena s myšlenkou, že poběží ve vlastním vykreslovacím vlákně, a proto způsob, jaký se s ní pracuje, nabádá ke sdružování prvků do větších celků.

Protože se ve třídě `VirtualMicContainer` vyskytují editovatelné textové prvky, je její úlohou změny v textovém poli registrovat a adekvátně s nimi naložit. Pokud se textové pole změní, zavolá funkci `valueChanged` této třídy, která provede jednoduchou úpravu vstupního textu. V tomto případě je pouze kontrolní část požádána o transformaci znakových řetězců do hodnoty parametru. Tato hodnota je následně poslána kontrolní části jako nová hodnota parametru.

I přesto, že celá třída knihovna VSTGLUI je vytvořena pro účely VirtualMic, je v souboru `VirtualMicGUI` k nalezení i implementace přepínatelného tlačítka, které tuto knihovnu využívá a rozšiřuje ji. Tlačítko slouží k tomu, aby přepínalo tři stavy: vypnuto, zapnuto a zrcadlový mód. Je založeno na stejném principu jako zbytek VSTGLUI, a proto bude tento princip popsán v odpovídající kapitole 7.5.

Všechny prvky virtuální jednotky jsou uloženy ve třídě `VirtualMicContainer`. Tato třída je udržuje aktuální a ve styku se zbytkem zásuvného modulu. Tím je také vytvořena celá pravá strana editačního okna. Levá strana je vytvořena opět ve třídě `VirtualMicEditorView` po vytvoření čtyř kontejnerů. Je zde vytvořeno okno, které taktéž využívá pokročilejší funkce OpenGL pro vykreslení 3D prostoru. A je toho docíleno kombinací knihoven VST3D a VSTGLUI. Třída `VSTGLUIView` si bere jako vstupní parametr objekt třídy `Scene`, konkrétně třídu `VirtualMicScene`, ve které je specifikováno zobrazení jednotlivých prostorových směrových charakteristik virtuálních mikrofonů.

V následujících podkapitolách bude podrobně popsána funkce a princip obou knihoven VST3D a VSTGLUI a jejich začlenění do zásuvného modulu VirtualMic.

7.4 Knihovna VST3D

Pro potřeby zobrazení 3D objektů uvnitř editačního okna byla v rámci této práce vytvořena jednoduchá knihovna VST3D. Její těžiště spočívá ve třídě `CGObject`, která uchovává informace o pozici, natočení a relativní velikosti 3D objektu - vše pomocí podpůrné matematické knihovny glm. Na základě těchto informací je možné vypočítat *modelovou matici* řádu 4x4. Ta je potřebná při vykreslování 3D objektu k pozicování jednotlivých dílčích bodů – vertexů, od počátku souřadnic, tak aby grafická procesní jednotka prováděla co nejméně operací.

Modelová matice je uvnitř externí knihovny glm vypočítávána vynásobením *velikostní matice* (scaling matrix), *rotační matice* (rotation matrix) a *posuvné matice* (translation matrix) v uvedeném pořadí.

Velikostní matice S je jednoduché vytvořit zadáním hodnot do hlavní diagonály 4x4 matice. Matice vypadá takto

$$\mathbf{S} = \begin{bmatrix} X & 0 & 0 & 0 \\ 0 & Y & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

kde proměnná X představuje zvětšení v ose x, Y zvětšení v ose y a Z zvětšení v ose z. Pokud jsou hodnoty těchto proměnných větší jak 1, bude objekt zvětšen v příslušné

ose, pokud menší, bude zmenšen. Pokud bude hodnota rovna jedné, objekt zůstane stejně veliký.

Rotační matici lze realizovat několika způsoby. V této práci bude uveden pouze algoritmicky výhodný výpočet ze zdroje [17], kde jednotlivé hodnoty sinů a kosinů jsou předem vypočítány. Pokud je dáno, že

$$\begin{aligned} A &= \cos(\alpha) \\ B &= \sin(\alpha) \\ C &= \cos(\beta) \\ D &= \sin(\beta) \\ E &= \cos(\gamma) \\ F &= \sin(\gamma) \end{aligned}, \tag{7}$$

pak je rotační matice $\mathbf{R}$ rovna

$$\mathbf{R} = \begin{bmatrix} CE & -CF & D & 0 \\ BDE + AF & -BDF + AE & -BC & 0 \\ -ADE + BF & ADF + BE & AC & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{8}$$

kde α je natočení v ose x, β natočení v ose y, γ natočení v ose z.

Posuvná matice je definována rovnicí 9

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{9}$$

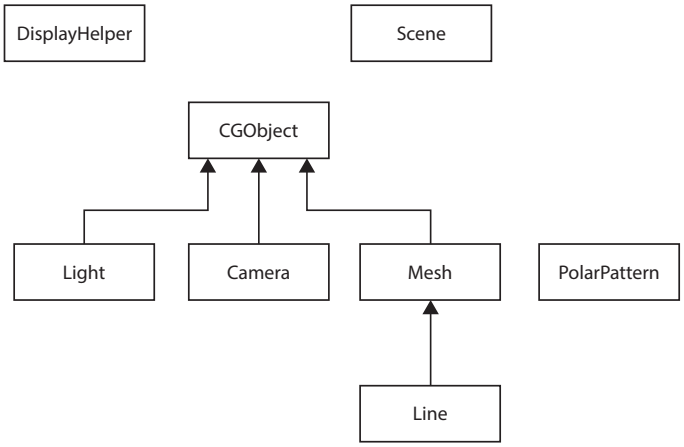
kde X je posunutí v ose X, Y posunutí v ose Y a Z posunutí v ose Z. Vynásobením této matice s vektorem aktuální pozice je dosaženo posunutí pozice o dané hodnoty posunu.

Vynásobením těchto matic dohromady (viz vzorec 10) v daném pořadí dojde k vytvoření nové – modelové matice. Jejím vynásobením s každým vektorem pozice bodu objektu dojde k příslušnému zvětšení, otočení a posunutí vzhledem k počátku souřadnic.

$$\mathbf{M} = \mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S} \tag{10}$$

Obdobou modelové matice je pohledová matice, která určuje pozici a natočení virtuální kamery. A pozicuje jednotlivé body objektu, aby zohledňovaly bod, ze kterého je na ně nahlíženo.

Z třídy `CGObject` dědí třídy `Mesh`, `Camera` a `Light`. Třída `Mesh` nabízí úložné možnosti pro uchování informací o tvaru, texturových souřadnicích a normálách jednotlivých vertexů objektu. Také je zde k nalezení funkce `Render`, která přijímá jako parametry ukazatele na kameru a světlo ve scéně. Aby objekt mohl být ve scéně



Obrázek 5: Třídy knihovny VST3D

správně vykreslen s perspektivní projekcí, musí program grafického akcelerátoru znát pozici a vlastnosti kamery, a pro správné barevné vykreslení i pozici světla.

Tyto informace jsou uloženy v třídách `Light` a `Camera`. Třída `Light` rodičovskou třídu `CGObject` neobohacuje prakticky o žádný z parametrů, je zde ale připravena pro budoucí rozšíření. Naopak třída `Camera` obsahuje informace o zorném úhlu kamery a velikostním poměru mezi šířkou a výškou zorného obrazce. Z těchto údajů zvládne matematická knihovna glm vypočítat perspektivní matici, jejíž vynásobením s každým bodem 3D objektu vznikne dojem prostorového vnímání – vzdálenější body se budou nacházet blíže u sebe a bližší dále od sebe.

V projektu VirtualMic ale třída `Mesh` sama o sobě využita nebyla, pouze její potomek třída `Line`, která obsahuje informace o začátku, konci a barvě úsečky. Vykreslovací funkce této třídy – `Render` obsahuje triviální instrukce pro OpenGL pro vykreslení jednoduché čáry. Protože se úsečky mají nacházet taktéž v perspektivní projekci, jsou oba body upraveny ve Vertex shaderu *transformační maticí* **MVP**.

Transformační matice se skládá z modelové matice **M** úsečky, z pohledové matice **V** (zohledňující pozici kamery) a z projekční matice **P** (zohledňující principy perspektivy). Výsledná pozice v rámci OpenGL kontextu je dosažena vynásobením této matice se vstupní informací o pozici jednotlivých bodů.

Shadery jsou pro úsečku definovány přímo ve zdrojovém souboru. Jedná se pouze o textovou informaci, která je vždy při otevření editačního okna kompilována a svázána dohromady s ostatními shadery v grafické programy. Tento způsob se při práci s OpenGL často opakuje, proto pro tyto účely byla do knihovny VST3D vytvořena statická třída `DisplayHelper`. Její hlavní silou je vytváření shaderů z řetězců kódu GLSL. Shaderu je při vytváření nejprve přiděleno identifikační číslo. K tomuto číslu je do paměti grafické karty přiřazen znakový řetězec kódu, poznámka o typu shaderu, a posléze je tento kód zkompilován. Pokud je shader zkompilován bez problémů, je možné ho začlenit do grafického programu.

To zvládne obstarat také třída `DisplayHelper` funkcí `CreateProgram`. Její vstupním parametrem je série identifikačních čísel shaderů, mezi kterými má proběhnout spojení. Při úspěšném navázání je možné všechny použité shadery smazat, nacházejí se již v programu. Mezi další funkce `DisplayHelperu` patří také nahrávání OBJ souborů pro konstrukci třídy `Mesh`, ale také nahrávání kódu glsl z externích zdrojových souborů.

Všechny popsané operace jsou realizovány funkcemi OpenGL, které prostředkují komunikaci mezi grafickým akcelerátorem a většinou funkcí knihovny VST3D. Funkce OpenGL lze snadno rozpoznat písmeny gl na začátku jejich názvu. Mají ovšem omezení, že smí být volány pouze tehdy, pokud je OpenGL kontext uzamknut pro používání.

Velmi podobnou funkčnost jako třída `Mesh` má třída `PolarPattern`. Tato třída byla začleněna do knihovny speciálně pro potřeby zásuvného modulu VirtualMic a jejím cílem je vytvořit 3D objekty pro směrové citlivostní charakteristiky virtuálních mikrofonů.

Třída `PolarPattern` je vytvářena se vstupní hodnotou hustoty vertexů základního objektu, ze kterého budou vytvářeny směrové charakteristiky – koule. Na začátku konstruktoru jsou vytvořeny body se stejnou vzdáleností od počátku souřadnic a následně uloženy do paměti grafické karty. Taktéž jsou zde vytvořeny shadery, které budou tyto body za běhu zásuvného modulu upravovat. Shadery jsou pro přehlednost umístěny ve speciálních souborech `polarPatterns.vert`, `polarPatterns.geom` a `polarPatterns.frag`. Jejich funkce bude popsána níže.

Vertex shader zvláštní funkci nemá, pouze přeposílá pozice bodů geometry shaderu. Jeho začlenění do programu je ale nutné, neb se jedná o vstupní místo pro veškeré informace o pozici vertexů. Hlavní práci provádí geometry shader. Ten je pro každý bod, který mu vertex shader pošle, volán čtyřikrát. Toho je docíleno pomocí definování klíčového slova `invocations = 4` na začátku shaderu. Shader je volán čtyřikrát pro každou ze směrových charakteristik virtuálních mikrofonů a využito je faktu, že každá z charakteristik má stejné vstupní vertexy, ale pouze jiný tvar a jiné natočení.

Změna tvaru z charakteristiky mikrofonů nultého řádu do charakteristiky kombinovaného mikrofonu probíhá ve funkci `main` podle rovnice 1, kde je využito skutečnosti, že pozice v ose z je taktéž kosinovou funkcí úhlu vyzařování v určitém směru. (viz kód 2) Jelikož je původní vektor pozice zarovnán do koule, jeho vynásobením se skalární proměnnou `gain` vzniká nová pozice bodu.

Zdrojový kód 2: Výpočet citlivosti směrové charakteristiky.

```
gain = abs(dir[unit] + (1 - dir[unit]) * gl_in[0].gl_Position.z);
```

Tím je dosaženo správného tvaru směrové charakteristiky; zbytek shaderu se stará o její správné natočení (pomocí transformační matice **MVP**) a výpočty pro další článek grafického programu – fragment shader.

Nutno ještě zmínit, že pokud je virtuální mikrofon, pro který je směrová charakteristika vypočítávána, v módu zrcadlového zobrazení, je aktuální bod ještě zrcadlově zduplikován v ose z. Výstupem geometry shaderu totiž může být více bodů, než se nacházelo na vstupu.

Ve fragment shaderu probíhá pro každý vstupní bod výpočet jeho barvy. Je zde velmi jednoduché stínování v reakci na vstupní hodnotu pozice světla. To je realizováno pomocí základních znalostí zákonů odrazu. Protože informací, se kterou pracuje grafický akcelerátor, je pouze umístění bodu, je nutné specifikovat teoretickou normálu, jakým směrem je bod natočený.

V případě třídy `PolarPattern` se normála nevypočítává, využije se pouze faktu, že základní tvar koule obsahuje body, jež mají od středu stejnou orientaci jako jejich normála. Do fragment shaderu je tedy z geometry shaderu poslána tato hodnota transformovaná navíc ještě pohledovou maticí. Dochází tomu tak i v případě, že geometry shaderem byl změněn tvar, a normála tedy neodpovídá směru, jaký by měla doopravdy mít. Tato nesrovnalost je řešitelná samostatným výpočtem normály; pro potřeby zásuvného modulu VirtualMic je ale tenhle způsob implementace dostačující.

Ve fragment shaderu jsou pak vypočítány hodnoty kosinu úhlu, jež svírají vektory světla a normály (cosTheta) a kosinu úhlu vektoru odraženého světla a směru natočení kamery (cosAlpha). Tyto hodnoty pak slouží jako činitelé pro výslednou barvu. Barva je vypočítávaná ze tří barevných složek: ambientní, rozptylové a odrazové (specular). Výsledné hodnoty barev jsou ještě vyděleny kvadrátem vzdálenosti bodu od světla, takže světlu vzdálené body jsou obarveny pouze ambientní barvou.

Zdrojový kód 3: Výpočet barvy uvnitř fragment shaderu.

```
color =
MaterialAmbientColor +
MaterialDiffuseColor * LC * LP * cosTheta / pow(distance,2) +
MaterialSpecularColor * LC * LP * pow(cosAlpha,5) / pow(distance,2);
```

Takto definovaným shaderům je potřeba vytvořit i místa v paměti, odkud mají být vstupní informace čerpány. To je provedeno sérií volání funkcí pro alokování místa v paměti `glGetUniformLocation` v konstruktoru třídy `PolarPattern`.

Tato místa v paměti jsou pak naplněna aktuální informací pokaždé, když je zavolána funkce Render. Aby byla místa naplněna pro správný shader, je nutné vždy zavolat funkci glUseProgram s identifikačním číslem aktuálního programu. Ve funkci `Render` jsou nejen připraveny transformační a pohledové matice, ale také vyvolán start grafického programu funkcí `glDrawElements`.

Třída `PolarPatterns` také obsahuje funkci `update`, která po zavolání vždy uvede lokální hodnoty parametrů do aktuálního stavu.

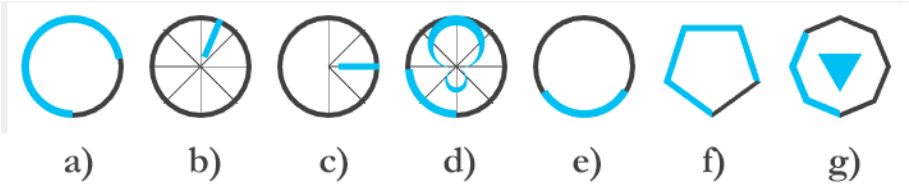
V popisu knihovny VST3D bylo tedy zmíněno, že pro vykreslení 3D objektů na obrazovku je potřeba znát údaje o objektu (ve třídách `Mesh` nebo `PolarPattern`), o světle (`Light`), ale také o pozici a nastavení kamery (třída `Camera`). Je vhodné objekty těchto tříd soustředit do větších celků – scén. Od toho je v knihovně VST3D poslední virtuální třída `Scene`.

Třída `Scene` kromě hlavní kamery a jednoho světla obsahuje seznam všech 3D objektů a informaci o obdélníku, do kterého se scéna má vykreslovat. Taktéž jsou zde připraveny funkce, které mají být volány, pokud dojde k pohybu nebo kliknutí myši.

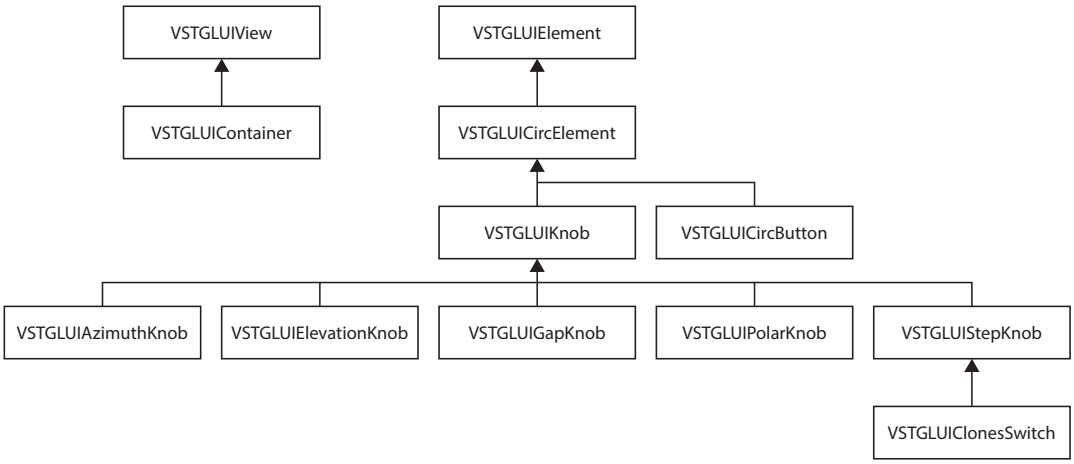
Tato virtuální třída je v projektu VirtualMic využita ve třídě `virtualMicScene`. Je zde inicializována výše popsána třída `PolarPattern` a také vytvořeny čáry znázorňující osy systému souřadnic pomocí tříd `Line`. Tato scéna také definuje pohyb kamery při interakci uživatele prostřednictvím táhnutí myši tak, aby se kamera pohybovala okolo vizuálních podob směrových charakteristik.

7.5 Knihovna VSTGLUI

Knihovna VSTGLUI se zaměřuje výhradně na realizaci grafického uživatelského rozhraní. Využívá plně OpenGL, ale pouze k vykreslování 2D grafiky. Hlavní myšlenka knihovny je posílat do grafické karty co nejméně informací. Pro sérii točítek, které byly pro knihovnu VSTGLUI vytvořeny, je charakteristické to, že základní vstupní informací je poloha středu točítka a poloměr točítka. Samozřejmě jsou s každým voláním vykreslovací funkce draw posílány do grafické paměti i informace o hodnotě nastavení, ale těchto informací je minimální množství.



Obrázek 6: Předdefinované uživatelské prvky knihovny VSTGLUI.



Obrázek 7: Dědičnost tříd knihovny VSTGLUI

Vizuální podobu točítka specifikuje grafický program, nejčastěji geometry shader, který kolem pozice středu vytvoří množství na sebe navazujících trojúhelníků. Možnosti geometry shaderu dovolují vytvářet další primitivní objekty (čáry a trojúhelníky) do souvislého pásu (stripu). To znemožňuje vytvořit složitější strukturu využívající popředí i pozadí. Proto je každé z točítek tvořeno často více jak jedním grafickým programem.

Třídy z této knihovny lze rozdělit do dvou skupin – na spravující elementy a na spravované elementy. Mezi spravující elementy patří hlavně třída `VSTGLUIView`, která má na starosti základní nastavení OpenGL kontextu pro specifickou oblast obrazovky. Přijímá vlastnosti z již zmiňované třídy VSTGUI `COpenGLView` a je vytvořena při otevření okna stejně jako veškeré ostatní uživatelské prvky. Hlavní funkcí této třídy je vykreslovat scény knihovny VST3D a je tak realizována levá polovina editačního okna zásuvného modulu VirtualMic.

Protože nastavení OpenGL pomocí VSTGUI neumožňuje přesně určit počet vzorků pro využití multisamplingu jako hlavní metody anti-aliasingu, bylo při tvorbě knihovny VSTGLUI nutno přistoupit k jiné metodě zlepšení estetické stránky vykreslení. Tato zdánlivě nepodstatná věc, aby hrany vykreslených prvků nebyly „zubaté“, hraje důležitou roli v otázce, proč OpenGL vůbec implementovat. OpenGL je použito hlavně z důvodů vizualizovat interaktivně zvukové parametry bez nutnosti prvky dopředu vypočítávat, jak je tomu ve většině dnešních zvukových zásuvných modulů.

Anti-aliasingu zde tedy bylo dosaženo pomocí využití pomocného framebufferu a renderbufferu. Ve stručnosti se jedná o nastavení celého vykreslovacího systému, aby se místo vykreslování do kontextu editačního okna vykreslovaly grafické prvky do textury. Tato textura (obrázek) je pak jednoduše vykreslena na patřičná místa grafického kontextu. Při nastavení tohoto samo o sobě zbytečného kroku je však možné zvolit metodu pro docílení hladkých hran. V případě knihovny VSTGLUI se využívá metody multisamplingu s osmi vzorky.

Rozšířením třídy `VSTGLUIView` je `VSTGLUIContainer`, která jako spravující element uchovává uživatelské prvky (např. točítka), které musí dědit ze základní třídy `VSTGLUIElement`. Třída se při otevření editačního okna inicializuje a při každém potřebném volání i vykresluje do prosotru, který jí byl svěřen.

Protože všechny uživatelské prvky využité v pluginu VirtualMic mají charakter kružnice, bylo nutno změnit způsob, jakým zásuvný modul vyhodnocuje informaci, zda se kurzor myši nachází nad uživatelským prvkem. To je definováno ve třídě `VSTGLUICircElement`, kde je počítána vzdálenost myši od centra točítka, a v případě, že je tato vzdálenost menší než poloměr, tak je zapsáno do proměnné `hover`, že se myš nachází nad uživatelským prvkem. Tato proměnná je důležitá pro další funkce, které reagují na klik myši, a také pro vizuální podobu prvku, kdy je nutné, aby koncový uživatel viděl, že editační okno zásuvného modulu reaguje na jeho podněty.

Všechny další třídy nacházející se v knihovně VSTGLUI dědí z třídy pomocné `VSTGLUICircElement`. Základním jednoduchým otočným elementem simulující otoč-

ný potenciometr je třída `VSTGLUIKnob` (obr. 6a), která umožňuje jednoduchou změnu parametru. Normalizovaná hodnota parametru je zde znázorněna barevnou oblastí jinak šedé kružnice.

Asi pouze v zásuvných modulech pro tvarování směrových charakteristik nalezne využití třída `VSTGLUIPolarKnob` (obr. 6d), která hodnotu kromě výseče kružnice znázorňuje i půdorysnou vizualizací směrové úhlové charakteristiky virtuálního mikrofonu.

Pro skokové změny parametru je zde třída `VSTGLUIStepKnob` (obr. 6f), která svým tvarem znázorňuje počet možných hodnot. Pro určování úhlů elevace a azimutu v rámci modulu VirtualMic zde existuje dvojice tříd `VSTGLUIElevationKnob` (obr. 6c) a `VSTGLUIAzimuthKnob` (obr. 6b). Jejich vlastnost se však nijak výrazně neliší od třídy `VSTGLUIKnob`, rozdíl je pouze vizuální. Zbylé třídy jsou do knihovny implementovány pro možné další rozšíření zvukového modulu VirtualMic (obr. 6e a 6g).

Všechny využité uživatelské prvky jsou v pluginu VirtualMic implementovány po skupinách. Každá skupina upravuje jednu virtuální jednotku a je realizována objektem `VSTGLUIContainer`, který se nachází ve třídě `VirtualMicContainer`, jak již bylo zmíněno výše.

7.6 Popis použíti pluginu VirtualMic

Použití pluginu VirtualMic je vzhledem k počtu vstupů i výstupů komplikovanější, než bývá zvykem u ostatních zásuvných modulů. Z tohoto důvodu bude použití pluginu věnována tato kapitola, a to konkrétně v hostitelské aplikaci Reaper, která dovoluje upravovat toky signálů z a do jednotlivých zvukových sběrnic a stop.

Pro správný chod zásuvného modulu je nutné poskytnout čtyři vstupní signály, které byly zachyceny technikami označované v ambisonii jako A-Format nebo B-Format. Tyto vstupní signály jsou nejčastěji v podobě zvukové nahrávky. Ukázkové zvukové záznamy lze nalézt na internetu v podobě čtyřkanálového souboru s příponou `.aif`. Soubor lze pak vložit do jedné stopy, ve kterém se také nachází zvukový zásuvný modul VirtualMic. V případě vlastního záznamu je vhodné jednotlivé stopy čtyř mikrofonů vložit do čtyř samostatných stop a bude tak učiněno i v následujícím popisu použití. Při absenci vhodných nahrávek je pro testování postačující do stop umístit generátory tónů s rozdílnými frekvencemi.

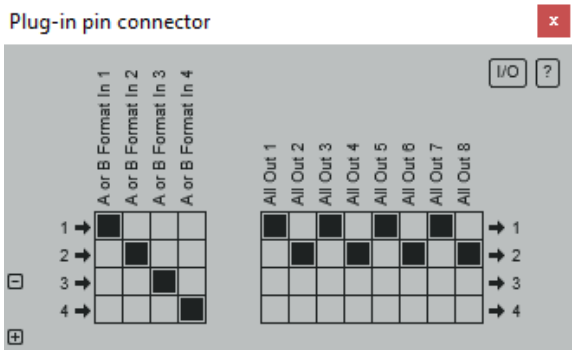
Dalším krokem je vytvoření stopy, která bude obsahovat zásuvný modul VirtualMic. Protože v tuto chvíli je výstup všech stop směřován do hlavní „Master“ sběrnice, nebude funkčnost zásuvného modulu zcela zřejmá. To lze vypnut tlačítkem *Master send*. Poté je nutné vstupní stopy nasměrovat do stopy zásuvného modulu pomocí tlačítka Route pro každou stopu zvlášť. V nově otevřeném „routovacím“ editačním okně lze přesměrovat signál pomocí nastavení funkce *Add new send* s cílem kanálů 1 až 4 stopy VirtualMic.

Pro B-Format by směrování vstupních signálů mělo být následující. Na kanál 1 by měl připadnout signál všesměrového mikrofonu, označovaného také jako W mi-

krofonu. Kanál 2 je připraven pro mikrofon 1. řádu směřující dopředu (a dozadu), taktéž označovaný jako X mikrofon. Kanál 3 obsluhuje vstup levého a pravého směru a kanál 4 slouží pro vertikální osu, tedy směr „nahoru a dolů“.

Pro A-Format jsou kanály vytvořeny pro mikrofony v tomto pořadí: LB, LF, RF a RB podle označení mikrofonů mikrofonní sestavy A-Format.

Nyní by již měla fungovat veškerá nastavení provedená v editačním okně zvukového zásuvného modulu. Avšak pouze pro první mikrofonní jednotku, jejíž výstupem jsou výstupní kanály 1 a 2, které jako jediné směřují do hlavní sběrnice. Pro ostatní mikrofonní jednotky je potřeba „přeroutovat“ výstupy kanálů 3-8 do dalších pomocných stop, nebo využít „routovací matice“ v okně zásuvných modulů. (viz obrázek 8)



Obrázek 8: Routovací matice pro stereofonní poslech v programu Reaper

Každý z parametrů lze nastavit buď zadáním konkrétní hodnoty kliknutím na číslo pod točítkem nebo otočením točítka do požadované pozice. 3D pohled lze měnit tažením myši v jeho regionu. Přepínání mezi A-Formatem a B-Formatem lze nalézt v levém horním rohu vizualizačního okna.

8 Závěr

V této bakalářské práci byly popsány principy zvukových zásuvných modulů pro zpracování zvukového signálu. Jako zástupce prostředí zvukových zásuvných modulů bylo zvoleno prostředí *VST*. Struktura této technologie byla rozebrána s důrazem na řešení grafického uživatelského prostředí.

Taktéž byl shrnut princip tvorby grafických aplikací využívajících technologie *OpenGL* a popsán jazyk *GLSL*, ve kterém jsou psány grafické programy. S ohledem na tuto teorii byl prozkoumán způsob, jak technologii OpenGL implementovat do zvukového zásuvného modulu VST.

V další části práce je popsán pojem tvarování směrových charakteristik a jejich využití v odvětví záznamu prostorového zvuku – v *ambisonii*. Zmíněny jsou zde základní dvě mikrofonové konfigurace *B-Format* a *A-Format*, které systému poskytují 4 vstupní signály. Jejich kombinací lze vytvořit tzv. *virtuální mikrofony*, které kopírují vlastnosti reálného mikrofonu – jeho natočení a směrovou charakteristiku.

Pro praktickou část bakalářské práce byl zvolen způsob implementace s pomocí knihovny *VSTGUI*, která je součástí vývojového balíku prostředí VST. Byl vytvořen zvukový zásuvný modul **VirtualMic**, který dovoluje vytvořit až osm virtuálních mikrofonů. VirtualMic dovoluje směrovat virtuální mikrofony všemi směry a měnit jejich směrovou charakteristiku. Výstupem je vícekanálový zvuk, který zohledňuje uživatelské nastavení zvukového zásuvného modulu.

Důraz byl kladen na vizualizaci virtuálních mikrofonů do prostorového zobrazení. K tomu byla v rámci této práce vytvořena knihovna **VST3D**, která dovoluje využívat grafického akcelerátoru k vykreslení 3D scény do editačního okna zvukového zásuvného modulu.

Pro plné využití potenciálu OpenGL byla vytvořena i knihovna **VSTGLUI**, která využívá tuto technologii k vykreslování grafických uživatelských prvků. Uživatelské prvky dovolují měnit hodnoty parametrů zvukového zásuvného modulu a adekvátně zobrazovat jejich aktuální hodnotu.

V rámci bakalářské práce byly splněny všechny cíle zadání. Potvrdilo se, že je možné využít grafického akcelerátoru pro zobrazení editačního okna zvukového zásuvného modulu.

Reference

[1] VLACHÝ, Václav. *Praxe zvukové techniky*. 3., aktualiz. a dopl. vyd. Praha: Muzikus, c2008. ISBN 978-80-86253-46-6.

[2] VST3 SDK Documentation. *Steinberg Website* [online]. Hamburg: Steinberg Media Technologies, 2016 [cit. 2016-10-22]. Dostupné z: `http://www.steinberg.net/en/company/developers.html`

[3] The Steinberg Story. *Steinberg Website* [online]. Hamburg: Steinberg Media Technologies, 2016 [cit. 2016-10-08]. Dostupné z: `http://www.steinberg.net/en/company/aboutsteinberg.html`

[4] *Desktop Window Manager*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2016-10-10]. Dostupné z: `https://en.wikipedia.org/wiki/Desktop_Window_Manager`

[5] SEGAL, Mark a Kurt AKELEY. *The OpenGL Graphics System: A Specification Version 1.0*[online]. Sunnyvale: Silicon Graphics, 1994 [cit. 2016-12-04]. Dostupné z: `https://www.opengl.org/registry/doc/glspec10.pdf`

[6] *History of OpenGL* [online]. Beaverton: Khronos Group, 2015 [cit. 2016-12-10]. Dostupné z: `https://www.opengl.org/wiki/History_of_OpenGL`

[7] *Khronos Releases Vulkan 1.0 Specification* [online]. Beaverton: Khronos Group, 2016 [cit. 2016-12-10]. Dostupné z: `https://www.khronos.org/news/press/khronos-releases-vulkan-1-0-specification`

[8] *OpenGL Shading Language* [online]. Beaverton: Khronos Group, 2015 [cit. 2016-12-10]. Dostupné z: `https://www.opengl.org/wiki/Related_toolkits_and_APIs`

[9] *OpenGL: Related toolkits and APIs* [online]. Beaverton: Khronos Group, 2016 [cit. 2016-10-10]. Dostupné z: `https://www.opengl.org/wiki/OpenGL_Shading_Language`

[10] *OpenGL Loading Library* [online]. Beaverton: Khronos Group, 2016 [cit. 2016-10-10]. Dostupné z: `https://www.opengl.org/wiki/OpenGL_Loading_Library`

[11] OLSZTA, Pawel W., Andreas UMBACH a Steve BAKER. *FreeGLUT: The Free OpenGL Utility Toolkit* [online]. 1999. [cit. 2016-10-10]. Dostupné z: `http://freeglut.sourceforge.net/`

[12] GEELNARD, Marcus a BERGLUND Camilla. *GLFW - An OpenGL Library* [online]. 2002. [cit. 2016-10-10]. Dostupné z: `http://www.glfw.org/`

[13] QOpenGLWidget Class *Qt Documentation* [online].
Espoo: The Qt Company, 2016 [cit. 2016-12-03].
Dostupné z: `http://doc.qt.io/qt-5/qopenglwidget.html`

[14] OpenGLRenderer Class Reference *JUCE Documentation* [online].
Londýn: ROLI, 2016 [cit. 2016-12-03].
Dostupné z: `https://www.juce.com/doc/classOpenGLRenderer`

[15] HOCEVAR, Sam. Basic shading *Free tutorials for modern Opengl* [online]. 2004 [cit. 2016-07-30]. Dostupné z: `http://www.opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading/`

[16] SCHIMMEL, J. *Nové metody kódování a reprodukce prostorového zvuku.* Habilitační práce. Brno: Vysoké učení technické v Brně, 2016. s. 1-177.

[17] JUNGHANNS, Andreas, ed. *The Matrix and Quaternions FAQ* [online]. 2003 [cit. 2017-06-03]. Dostupné z: `https://www.j3d.org/matrix_faq/matrfaq_latest.html`

A Zásuvný modul VirtualMic a jeho zdrojové soubory

V přiloženém archivu (elektronická verze) popřípadě na přiloženém CD (tištěná verze) se nachází adresář *VirtualMic* obsahující plugin a jeho zdrojové soubory, adresář *ThirdParty* obsahující knihovny glew a glm. Obě knihovny mají přiložené jejich právní licence - modifikované MIT licence. Také je zde přiložen vývojářský balík VST3 SDK verze 3.6.5, který obsahuje licenční ujednání v hlavičkách všech souborů.

V adresáři VirtualMic je, kromě složky *trunk* se zdrojovými soubory a projektem připraveným ke kompilaci, také zkompilovaný soubor `VirtualMic.vst3`, který je potřeba, aby ho mohly hostitelské aplikace nalézt, umístit do složky `C:/Program Files (x86)/Common Files/VST3/`. Použití tohoto zásuvného modulu je rámcově sepsáno v kapitole 7.6.